# Availability Prediction Based on Multi-Context Data

PROJECT PLAN

Group: sdmay19-33

Client: Goce Trajcevski

Team members:
Justice Wright - Report Facilitator
Shane Impola – Scribe
Noah Chicchelly – Meeting and Communications Facilitator
Nick Schmidt – Software Systems Engineer
Tristan Anderson – Network Systems Engineer
Brendon McGehee – Hardware Systems Engineer

Team email: sdmay19-33@iastate.edu

Team Website: https://sdmay19-33.sd.ece.iastate.edu

Version 3.0
Revised: December 3, 2018

## Table of Contents

## List of Figures

## List of Tables

## List of Definitions

MySQL - My Structured Query Language
AWS - Amazon Web Services
VM - Virtual Machine

# 1 Introductory Material

## 1.1 Acknowledgement

The Availability Prediction team would like to thank Dr. Trajcevski for all of his technical and project advice given throughout the duration of this project.

## 1.2 Problem Statement

In broader terms, the Internet of Things (IoT) paradigm involves a seamless integration of heterogeneous entities from three different abstract settings: (1) sensing - i.e., detecting values of different physical phenomena of interest via different sensors; (2) networking and analytics - i.e., combining the heterogeneous data types, focusing on both transmission and analytics-oriented processing (with different aspects such as in-network vs. edge vs. cloud-based execution of algorithms); (3) decision and actuation - i.e., predicting a particular state in the future and taking a particular action (spanning from notification to orchestrating the operational mode of various devices, sometimes coinciding with the sensing ones).
In addition to the “main” sensed values, often times there are contextual dimensions that generate data that, under special (e.g., threshold-based) circumstances, will need to be incorporated in the workflow.

One application domain that is societally popular and could benefit from an IoT-based solution is the effective management of service in restaurants. In particular, waiting time is a parameter the estimation of which could benefit both the patrons (i.e., customers can plan their time/activities better) as well as the staff (i.e., better occupancy management can yield better profits). However, at the current state, such estimates are solely based on “experience” and, consequently, are: (1) highly unreliable; (2) not capable of making real-time adjustments based on data from multiple contexts. For example, it is not uncommon to be given an estimated wait time that is off by a factor of 15 mins or more[1]. Inaccuracy in such estimations leads to customer loss and increased dissatisfaction.

Our main task is to create an IoT based solution for improving the estimation of the wait-time in restaurants Towards that, we envision a system that will be based on an integration of the following three main modules:

1. Use of sensors to create a sensor network reflecting the occupancy in the restaurants (e.g., at the level of granularity of a seat).
2. The data from these sensors would then be aggregated to determine the state of the occupancy and will be relayed to a centralized server. Both periodic as well as event-based data will be catered.
3. We will then use this data to perform analytics that map a normalized dining experience and predict crucial event timepoints such as the time a table is likely ready for a check, the time a table is vacant, and the time it is bused and ready for new diners. This data

---

[1] https://www.quora.com/How-do-restaurant-hosts-estimate-wait-time

will be updated in real-time and be accessible to restaurant workers via a mobile app. This app will help more efficiently guide the workers time spent servicing active diners while simultaneously providing more accurate wait times to potential diners.

### 1.3 Operating Environment

The expected operating environment of this project will be integrated within the seating of restaurants. The system should not be exposed to any harsh weather conditions because it will remain inside in a regulated temperature environment. The only condition that the system might be exposed to is some dust and debris over a long period of time.

We will be design the hardware portion of the project to be minimally intrusive for the customers, so that their dining experience and quality of service are not affected. Due to this we expect that there should be no human interaction with the hardware.

### 1.4 Intended Users and Intended Uses

Our intended final project users are going to have a wide range of technical knowledge. The individuals that are going to be using this are both customers and employees of the restaurant that it will be implemented in. Although customers and employees will be using this in an app based form, the purpose of the app for each will be very different.

The employees will be using this app to input resturant data such as when the food has been given to the table, when the food has been removed from the table, and when the check has been given to the table. This information will then be used to better predict table wait time, which is what the customer use will be. The customer will use the app to see available tables as well as receive restaurant wait times.

### 1.5 Assumptions and Limitations

Assumptions:

- The final product will only be implemented indoors - the hardware of the project does not need to stand up to any harsh weather conditions.
- Hardware component will not have any severe limitations in terms of power supply, e.g.:
    - They will either have access to electrical outlets, or
    - A 12V power supply will available at the tables to power arduinos.
    -

Limitations:

- Sensors need to be unnoticeable inside a restaurant - The sensors used to collect data need to be implemented into a restaurant setting without impeding the normal operation of the restaurant.
- The application needs to be usable by all types of technical backgrounds - The app that we will be implementing in our project can not be overly difficult to use because people of all technical backgrounds will be utilizing the application.

### 1.6 Expected End Product and Other Deliverables

The final product of this project will include sensors with a microcontroller to collect data about tables, a networking component to organize and analyze the collected data, and an app component for either an employee or a customer. The components need to be able to communicate with each other but they do not need to communicate individually. The sensor component needs to be able to communicate with the networking component and the networking component needs to be able to communicate with the software component. A major milestone for our project is that all three components are effectively able to communicate with each other.

## 2 Proposed Approach and Statement of Work

### 2.1 Objective of the Task

The objective of the project is to design a product that will enable consumers and producers of restaurants to easily gather and access real-time data about seating availability in a specific restaurant. This system will consist of both individually placed sensor nodes for information gathering, as well as a mobile app  available to both customers and employees. The team goal is to implement the functionality required to have a usable app with this information. This will include hardware installed in seating areas and a fully developed server and database to communicate with our application.

### 2.2 Functional Requirements

The functional requirements will be split into customer use, and employee use. There will be two forms of the application for each of these users. Below is a list of the customer’s function requirements. The expected use case is to find a location you’re interested in visiting, check for generic wait times of that day, or click on more specifics for a top down view of available seating, or an estimated time of arrival for a given table.

- Choose a Location
- View an estimated wait time for that time of day
- View available seating
- View an estimated wait time for a specific table
- Real time updates on when tables become available (push notifications)

Employee use cases are going to be a lot of the same, but with a few extra perks. Employees will be able to see specific state’s of dining that a customer is in, as well as what was ordered:

- Choose a Location
- View an estimated wait time for that time of day
- View available seating
- View an estimated wait time for a specific table
- Real time updates on when tables become available (push notifications)

- View where a specific table is at in their meal (food ordered, food delivered, etc)
- View what was ordered at a specific table
- Override / shutdown specific functionality of the app (Table marked as out-of-order)

## 2.3 Constraints Considerations

### 2.3.1 Non-Functional Requirements

Availability - The app must be up 24/7 for estimated times at specific hours. Even if closed, information for future events is vital.

Data Integrity - Making sure our data is always accurate over the span of the app's lifetime. Allowing businesses and consumers the ability to trust the app and use it flawlessly.

Fault Tolerance - The app will continue to work if one system goes down. For example, if our live view of seating fails to work, you can still get an estimate of the wait time for that time of day.

Scalability - Restaurants are different shapes and sizes. The app must be able to accomodate a large or small number of seats, seating arrangements, and real-time data at smaller, or huge scales.

Usability - The app should be easily accessible by clients as well as employees. An intuitive way to view the information you want, immediately.

### 2.3.2 Standards

The standards of the project will be built with a few assumptions. For one, we will use MySQL that has standards built-in. However, with the data release of people, we will be working under the assumption that the data is of our own use. In the scope of the project, that is not one that we are going to be focusing on. Under reasonable circumstances we will be releasing data that is known by at least an employee of a restaurant, nothing that isn't "public" to the restaurant.

The customer use-cases of the app will also be limited to very generic information and ETAs, rather than giving specific data about what a table is or is not eating, or how much their bill is. This is trivial for standards and practices. Employers will not release that information, but have it at their use to better help customers with information they may request.

## 2.4 Previous Work and Literature

The app will resemble a mix up of a few different entities that are already in the market today. The first one is google:
Google takes information from restaurants to estimate the popular times of a restaurant. They do other things like general prices, reviews, etc. But in relation to our app, google only touches on one aspect, the estimated "popular times" of a given restaurant. Our app will have this same feature but implemented in a more specific way. Google uses the generic approach of taking

users locations from their phones and aggregating the data. This can make a statistic fluctuate if people are there to eat, eat for a long time, just buy something nearby, etc. Our app seaks to perfect this feature and make it more accurate.

The second thing on the market that our app might resemble is event seating services:

You can now buy tickets to specific seats for a large event, or movie, and see top down views of the seating arrangements with what seats are what price, and if they are taken or not. This is the same process we want, but for restaurants. Specifically for wait times on specific tables, or seeing how busy it is in real time, from information given from us, the restaurant. Rather than aggregated data from location pings of mobile phones.

We are not following previous work for this. This means we have to implement all aspects of functionality. Starting with hardware implementation to give us the data we require to implement our functional requirements within the app.

### 2.5 Proposed Design

We now provide a high-level overview of the proposed design and discuss the main components that will be involved.[2] Our design will be relatively simple.

1. Find a way to gather the required data (Are people at a table, where are they within their meal, etc.)
2. Communicate that data to a server and database for organization and availability
    a. We will have dynamic and static queries for both consumer and employees
    b. We will have the ability to speak to the hardware, as well as the front-end app.
3. Implement the front-end application for employees and customers to use in real-time.

Our scope is relatively well defined. There are a few design alternatives to use/non-use cases. And there could be widened scope by adding in security and data-release-standard requirements. These are all possibilities depending on how quickly things move throughout the Fall semester of 2018.

---

[2] The details of technology considerations are presented in Sec. 2.6

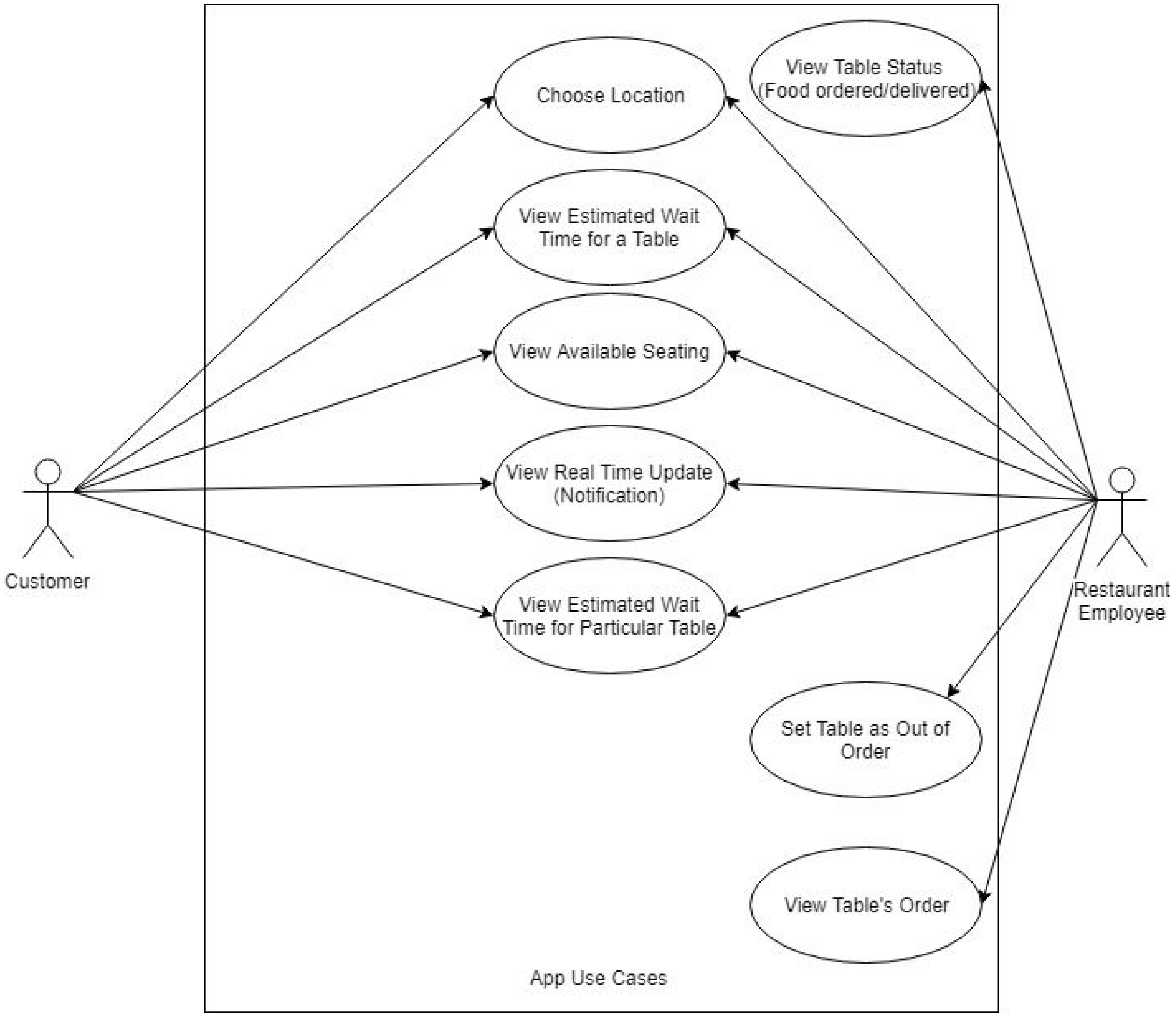

Figure 1: Use Case Diagram

| Name | Description |
|---|---|
| UC1- Choose Location | Choose a location from the list of available locations. |
| UC2- View Est. Wait Time for Table | View the wait time to receive any table at the given restaurant. |
| UC3- View Available Seating | View all of the seating for the restaurant and see which seats are available. |
| UC4- View Real Time Update | Receive a notification on the phone when a table becomes available. |
| UC5- View Est. Wait Time for | View the wait time of a specific table on the seating chart. |

| | |
|---|---|
| Specific Table | |
| UC6- View Table Status | View the status of the current table, whether it has ordered food, is currently eating food, has received check, ect. |
| UC7- Set Table as Out of Order | Set specific tables as out of order, making them unavailable for view in the app. |
| UC8- View Table’s Order | View the food that a specific table has ordered. |

Table 1: Use Cases

**Component Diagram**

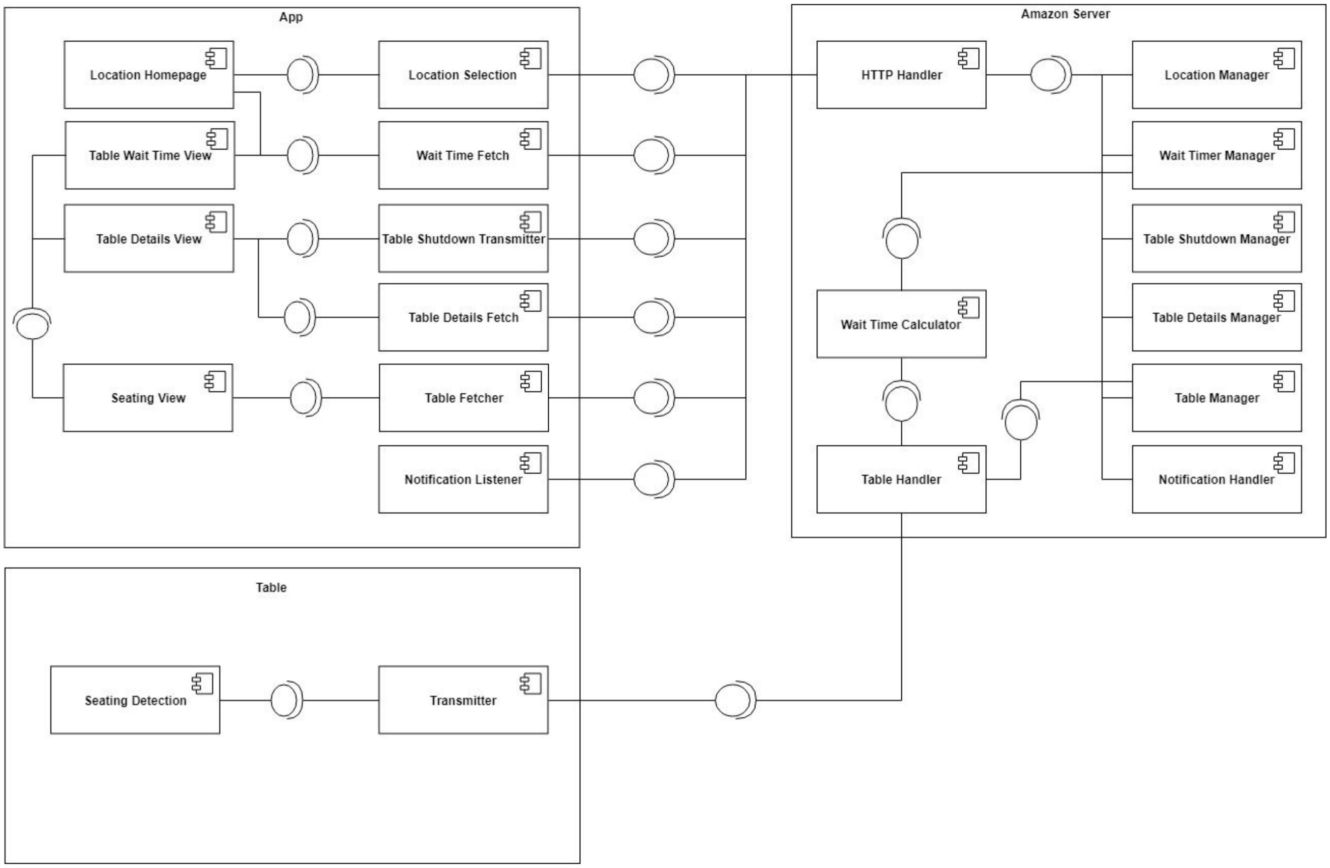

Figure 2: Component Diagram

Application:

| Component | Description |
|---|---|
| Location Homepage | The launch page of the app, will start with a location selection and update to the wait time for a table and navigation to other pages. |

| | |
|---|---|
| Location Selection | Will communicate with the server to get the available locations. |
| Table Wait Time View | Will hold the GUI and interactivity for seeing the wait time of a specific table. |
| Wait Time Fetch | Will communicate with the server to fetch the wait time for a table or restaurant. |
| Table Details View | Will hold the GUI and interactivity for viewing the details of a table (food ordered, how long the customers have been there, what stage they are in), will also contain GUI controls to set table as out of order. |
| Table Shutdown Transmitter | Will communicate with the server to notify the server when a table should be set as out of order. |
| Table Details Fetch | Will communicate with the server to retrieve the details for a given table to displayed in the details view. |
| Seating View | Will contain the GUI and interactivity for the seating of the restaurant. Will allow users to change into the wait time view, and if the user is an employee, the table details view. |
| Table Fetcher | Will fetch all of the information about tables available at the restaurant to be displayed in the app. |
| Notification Listener | Will listen to notifications from the server to notify the user when a table becomes available. |

Table 2: Application Component Description

Amazon Server:

| Component | Description |
|---|---|
| HTTP Handler | Will listen for HTTP requests and route them to the correct manager. |
| Location Manager | Will retrieve information related to the location. |
| Wait Time Manager | Will retrieve information related to the wait time. |
| Table Shutdown Manager | Will shutdown or open specific tables for the given restaurant. |
| Table Details Manager | Will retrieve detailed information about the tables (food ordered, current status). |
| Table Manager | Will retrieve information about the tables in the restaurant (locations, seating available, location). |

| | |
|---|---|
| Notification Handler | Will setup a notification line for the app, will notify the app when a table or a specified table becomes available. |
| Wait Time Calculator | Will calculate the wait time for a given table. |
| Table Handler | Will listen for transmissions from table transmitters and pass the data to the correct managers. |

Table 3: Amazon Server Component Description

Table:

| Component | Description |
|---|---|
| Seating Detection | Detects whether someone is currently sitting in the seat |
| Transmitter | Transmits relevant data. |

Table 4: Table Component Description

### 2.6 Technology Considerations

There are three essential subsystems to our overall design, each with their own technological considerations: the data collection system, the data analytics system, and the client-side application, all of which are pictured below.

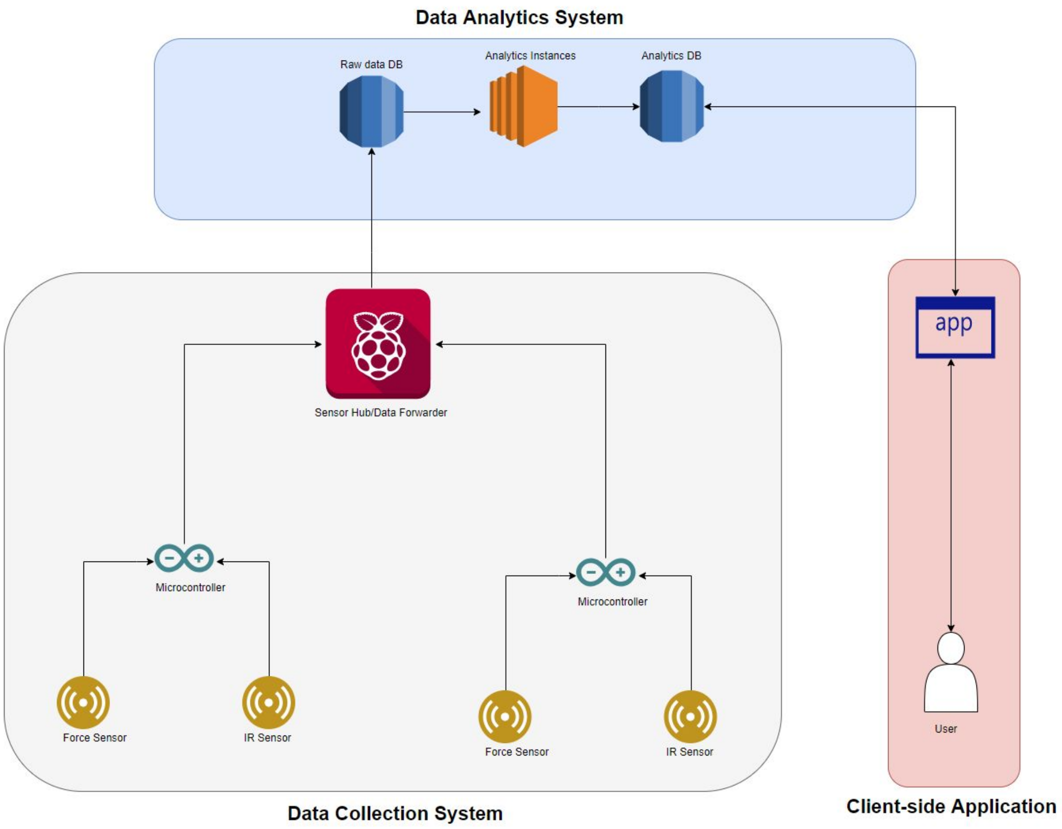

Figure 3: Technology Considerations

For the data collection system we needed multiple sensors to help determine occupancy of a table. Force sensors (load cells) were the original consideration, to be placed beneath the seats, thus triggering upon the sitting down of a guest. While this is a worthwhile datapoint it was determined further sensors would be necessary as both a precaution against false positives and a way to handle booth seating. Because of the nature of booth style seating, it is unsustainable to put enough force sensors in the seating area to accurately detect a seated occupant as weight is distributed more widely around a booth.

To overcome this shortcoming the decision was made to introduce IR sensors to the seating environment. IR sensors operate outside of the physical seat, which means they do not share the weakness of the load cells. In addition, seating is much more linear in a booth scenario, meaning IR blocking is much easier to achieve in such a scenario, so the addition of IR actually functions best in our previously worst visibility situation.

Both sensors are configurable to be controlled by a microcontroller which will both provide power and input necessary to the sensors as well as collect and transfer data from the sensors. In this instance we chose to use the Elegoo UNO R3 because we have team members who have experience using them, and they're the cheapest microcontroller commercially available with our desired chip, merging familiarity with lower production cost.

From here the sensor data needs to make its way to our data analytics clusters. We decided against outfitting each microcontroller with networking capability due to concerns over scalability concerns with both cost overloading our servers with too many connections in production. Instead we opted to aggregate the data from each microcontroller into a raspberry pi which bridges the data to our analytics system. This decision was made part to due accessibility since many team members already own and were willing to contribute pi's to the project as well as the price of a single pi with connectivity coming in at less price/unit than equipping each Uno that a pi could manage with wifi modules.

For the data analytics system we knew we needed multiple databases to store our sensor data both pre and post analytics as well as instances for performing the real-time analytics. This could be achieved in a multitude of ways, and the real decision here was where to host this system. Ultimately it was decided to use Amazon Web Services (AWS) for this system. This was decided because AWS, in addition to being the industry standard at this point, doesn't require us to operate or own our own hardware, and has free offerings we felt we would stay inside for the scope of this project making costs non-existent for this system. Additionally were our product to go into production AWS scales well if we ever escape the data usage of their free environment and offers many security tools and compliance aids that could be vital to expansion.

Now that we have our environment to host our analytics in, we required a means to access the end data and relay that information to our customers. Given that many restaurants already employ tablets in some form for their seating systems it was decided fairly early that some form of mobile application. The decision point here is what platforms will we support (iOS, Android) and what language/tools will we use during development. The decision to use Flutter was made for multiple reasons. Flutter is an open source project created by Google that is used to create mobile apps for both iOS and Android. The open source aspect means again that there will be no cost of creating the application, of which itself could eventually be sold to increase profitability of our system. The fact that both iOS and Android development are supported means that we can produce a product that should fit any companies existing infrastructure. Additionally Flutter uses Dart as its primary language, which while no one on our team has much direct experience with, is an object-oriented programming language similar to Java which our entire team has experience with.

## 2.7 Safety Considerations

The ideal testing scenario for our equipment is a mock restaurant booth where patrons are unaware they are even interacting with our product.  Because of this there become several safety considerations.

Sensors will need to be able to withstand and function under the weight of a fully grown adult. We rely on load cells to detect occupancy of seats, and a load cell breaking would both render our product useless, pose as a threat to our customer in the form of debris potentially puncturing their skin, as well as pose a fire risk if the wires were to come loose during the sensors destruction.  To ensure the safety of our customers care was taken to inspect the datasheet for max load capacity and compare that against our expected customers.  Additionally the load sensors would be put through numerous tests with clients of varying sizes before ever reaching production, and if a defect was discovered the product would not launch until an alternative sensor was found.  The connection integrity of the wiring would also be checked routinely during testing, and done by Brendon, who has the most experience with proper circuit configuration and maintenance.

Restaurant scenarios also introduce the possibility of food spills (both liquid and solid) being introduced into our system.  To reduce the risk of shock and failure the microcontrollers have been designed to be mounted outside of the typical dining area, adjacent to the booth where spills are very unlikely to occur.  All sensors have been designed to be placed within seats, tables, or other existing dining infrastructure, as to add a layer of abstraction between themselves and the dining surface, thus reducing the risk of direct contact.

## 2.8 Task Approach

Our approach to solving this problem will have three main components. The first component is the hardware, which will take the form of the various sensors that are embedded into the client environment. We need our sensors to be able to collect various bits on information about the occupants of a given table, so we need to find types of sensors that will be able to collect to most robust information possible. The data points that they generate should be able to be used in multiple predictions, and they should be relatively straightforward to implement. We decided to use both IR and force sensors, and pair each table with an Arduino Nano with internet connectivity to gather the data generated by the sensors at the table. This approach allows us to gather sensory information locally at each table before sending and relevant information directly to the server. There will be no need for a central computer or server for the restaurant to maintain in which all sensors will be connected. Ideally, this approach should be cheap and simple to install, while requiring minimal maintenance.

The second component of our solution are the web services. This is the part that will be doing all of the data aggregation, storage, analysis and prediction, and broadcasting of the processed information. We decided to use AWS for our web services because it meets our needs while being cloud-based, meaning clients won’t need to keep and maintain server hardware in-house. This also has the effect of reducing the initial cost of implementing our solution significantly.

There will be two components to our usage of AWS, the server and the database. The server will be used to aggregate any information sent to it by the sensors and use it to update the database. It will also periodically perform analysis on the contents of the database, making predictions that will be stored until the next one is made. The server will also handle data requests from employee and customer phones. The database will have relational tables that will store all the information from each sensor type and for each table, and an input table where unprocessed entries will be buffered. This approach allows us to make queries about information specific to one table, which might be more useful to guests, or for information about many different tables, which would be more useful to employees. The web services can be visualized in Figure 2.

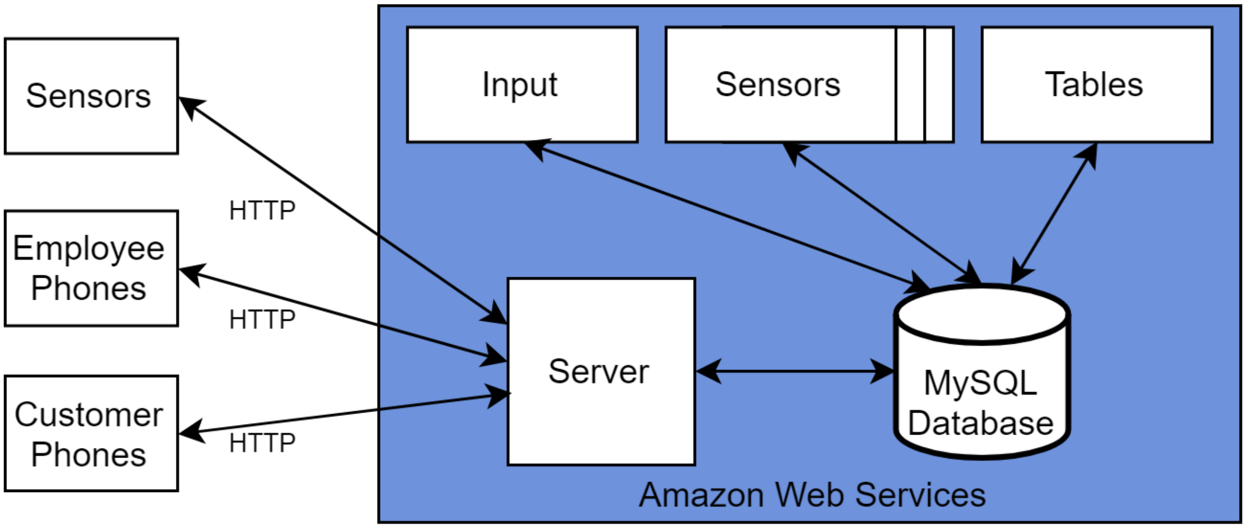

Figure 4: Task Approach

The third component of our solution is the user interface, the apps that employees and guests will use to get predictions. We need the user interface to be easily accessible to both types of users, so using mobile applications is what we've decided to do. The apps will need to be able to have updates pushed to them by the server and pull updates on demand. Using mobile apps also eliminates the need for restaurants to buy proprietary hardware for customer use, akin to the buzzers commonly seen today. Theft or damage of the user interface will no longer be a concern because customers will be using their own devices.

## 2.9 Possible Risks and Risk Management

Risk: There persists the possibility that our current sensor array won't provide enough data points to allow for accurate prediction analytics to be performed. As our weakest area of exposure is data analytics there's no one on our team that has a good enough grasp to rule this out.

Risk Level: Medium
Mitigation: Our design to use a series of microcontrollers to control and feed data from individual sensors leaves room for expansion. We are not currently using nearly every input port available to the controllers, and could add sensors to each system very easily. If the need for inputs ever exceeds our microcontroller we can simply scale up to a larger microcontroller.
Risk Level Post-Mitigation: Low

Risk: Our team has no experience with Dart or AWS which are cornerstones to the software side of our project. There lies a risk that this inexperience may be insurmountable and hinder progress and push back deadlines.
Risk level: Low-Medium
Mitigation: We have fallbacks that we are more familiar with for both Flutter and AWS if the learning curve starts to hinder our progress in a meaningful fashion. The ece department offers VM and database services that we have utilized for previous classes, and could use to replace AWS if need be, four team members have experience with these systems. Three team members also have experience coding directly for android using java and Android Studio if Flutter proves to be a choke point. Both of these replacement opportunities are highly viable and still are free offerings, meaning using these as fail-state fallbacks would only incur costs of time.

Risk Level Post-Mitigation: Low

## 2.10 Project Proposed Milestones and Evaluation Criteria

The first milestone would be to gather data from our sensor and process it through a microcontroller. This will be verified by configuring the controller to read and save data it reads from the sensor, purposefully triggering the sensor, and examining the data to ensure the sensor was triggered and the controller was aware the trigger.

The next milestone is to send this sensor data to a database for storage. To verify this, the sensor system consisting of the individual sensors, and microcontrollers will be hooked up to a raspberry pi, configured to transmit the sensor data to a database in AWS. We can then query that database in real time, and observe data being populated, verifying this milestone.

A third milestone will be configuring instances to consume the data in the database and perform predictive analytics with the data. To verify this we can load a dummy database with known data, and have our analytics instance perform analysis of this data. We can by hand determine the outputs we expect to see and confirm this milestone is achieved if the results match.

Another milestone will be interfacing with our web instances to observe the output of the analytics in a client-facing application. This can be verified by starting with a blank database, triggering the sensors, and verifying that the app updates corresponding to the presence of new data.

### 2.11 Project Tracking Procedures

Our team will be writing weekly status reports with descriptions from each team member on what they completed each week and also what they plan to complete the following week. Our team is also using Discord to quickly communicate project progress as well as ask other team members questions outside of face to face meetings. We are also using Trello to communicate goals of specific portions of the project as well a group portions of the project.

Along with our teams weekly status reports we also will have a weekly meeting with our client. In these meetings team members will present the progress that they made that week and also what they plan to complete the following week. Team members will also ask clarification questions about the current state of the project as well as the vision for the future of the project.

### 2.12 Expected Results and Validation

The final product should be able to give accurate predictions about how long customers will end up waiting until the next phase of their dining experience will begin, for example, how long they will need to wait before they are seated at a table. We will measure the success of our predictions based on the percent difference between the actual wait time and the expected wait time. Our goal is to have an average predicticted wait time be no greater than 10% away from the actual wait time. This metric will be our most important to work toward improving, as the success of our project will be based on this result.

Another result we’d like to produce is to determine the range of weights that we can use for calibrating the system. The range should be an ideal range to be inclusive of everyone that will be visiting the restaurant. This goal will be measured by the number of false positives per day that get reported or noticed by the waitstaff. Our goal is to keep this number below 2. We will test this by running the prediction algorithms on sets of test data during development, and listening to user feedback after deployment. A similar result to the last is discovering the optimal placement of sensors in seating areas of all shapes and varieties. This will be specific to the restaurant where the system is being deployed, and will require some testing in each variety of seating location to properly determine. The result of these efforts will again be measured by listening to customer feedback.

## 2.13 Test Plan

### Hardware Tests

**FR.1:** This is a test program for the an Arduino Nano that prints out sensor values in real time to test if the sensors function properly.

**Test Case:** Test if the sensors are outputting data as expected.

**Test Steps:**

1. Start the test program.
2. Manipulate the sensors by hand.
3. Watch the printed output and determine if they are as expected.

**Expected Results:** The expected printed output value should remain within 5% of the physical changes in the state of the sensor.

**FR.2:** This is test software that will take a set of raw input from the sensors and process it in order to determine if the arduino is processing sensor output properly.

**Test Case:** Test a set of data on both the arduino and a computer

**Test Steps:**

1. Run the arduino program, enabling a debug option that will send all inputs to the computer.
2. Simulate the data processing on the computer.
3. Compare the result of the arduino program to the simulation.

**Expected Results:** The output values from the arduino should be within 5% of the simulation.

### Server Tests

**FR.3:** This is a connectivity test that will ping the server and await a response to determine if the server is online and responding to requests.

**Test Case:** Test that the server responds to requests as expected.

**Test Steps:**

1. Send a request to the server test address.
2. Wait for a response.
3. Compare the response to a preloaded value.

**Expected Results:** The response from the server should match the preloaded value.

**FR.4:** This is a database test that will test the functionality of the database.

**Test Case:** Test that the database handles inputs and requests as expected.

**Test Steps:**

1. Send a request to the database test address.
2. Wait for the server to perform a set of automated queries on the database and respond with the result.

**Expected Results:** The result should indicate that the tests were successful.

## App Tests

**FR.5:** This is a set of test functions on the apps that implement the same functionality as described in tests FR.3 and FR.4 to determine if the app is communicating properly with the server. See those tests for details.

**FR.6:** This is a software test that will test the functionality of our predictions.

**Test Case:** Test the accuracy of our prediction algorithms against big data.

**Test Steps:**

1. Generate big data real-world simulations of our test environment
2. Cross the data with ours to make sure it's within an ~10% margin

## System Tests

**FR.7:** This is a general test to determine scope of our sensors.

**Test Case:** Test to determine the ideal weight range we should be sensing for.

**Test Steps:**

1. Test real-world data for a general range of weights that should be tested.
2. Verify we are only triggering at the minimum and maximum ranges.
3. Determine if we keep false positives below a threshold.
    a. If we do, test complete.
    b. If we do not, repeat and adjust min/max values.

**FR.8:** Environmental test to test if the system will work in a specific environment

**Test Case:** Test that the tables/chairs/location can house the required sensors.

**Test steps:**

1. Test units (tables/chairs) for the weight requirement for sensor trigger.
2. Test functionality of sensors in tested locations, repeat until sufficient.

**Expected Results:** This should result in sensor placement yielding results within 10% accuracy of the average.

# 3 Project Timeline, Estimated Resources, and Challenges

## 3.1 Project Timeline

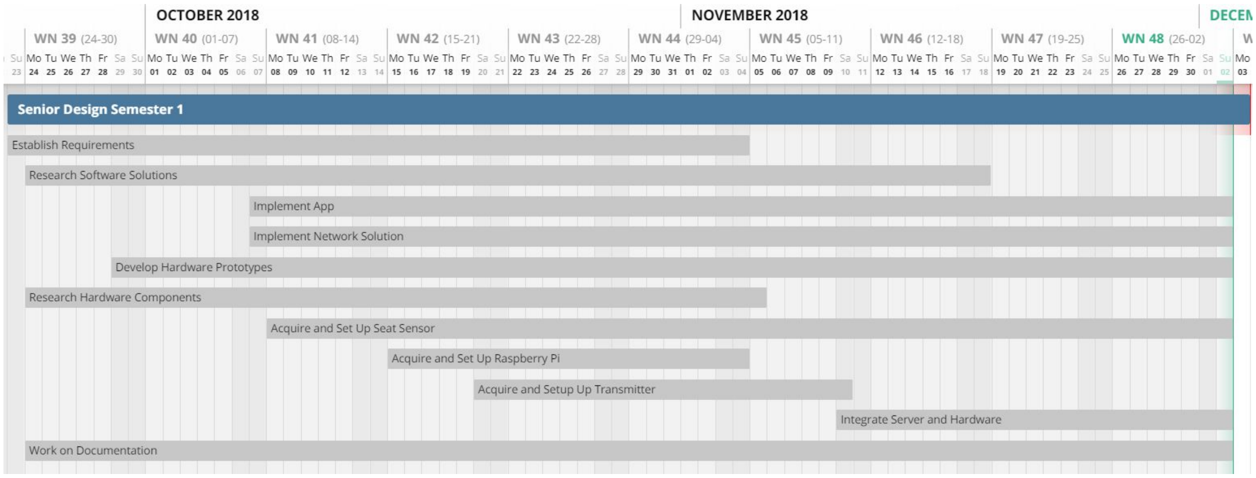

Table 5: Project Timeline Semester 1

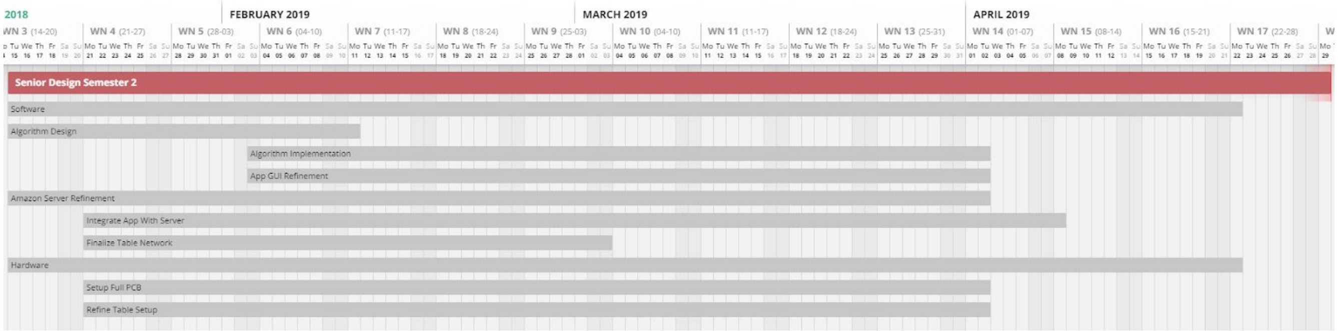

Table 6: Project Timeline Semester 2

The contributions for the project were transferred from Section 3.3 into a comprehensive plan for expected work requirements for the two semesters during which the project would be worked on. The expected time for each contribution was initially defined in the table as an estimate of the amount of man-hours it would take to complete among the team members assigned to completing it. Using our experience from past projects, we placed each contribution on the timeline roughly in the time range it would need to be completed by to continue to make progress toward the finished product. The expected time commitment that each member was expected to provide was used to determine the total length of each of the blocks on the timeline.

## 3.2 Feasibility Assessment

By the end of this project we expect to have an app capable of showing a live count of the current wait time for seating. We plan to include a version for both customers and owners. The customer version will show available tables and the live count. The owner version will contain detailed information and analytics about wait times at various times of the day and wait times for specific tables/foods.

On the hardware side of things we will have a system built to show a proof of concept. There will be at least one working table with a transmitter and seating sensors to show that the idea works. We will insure this design is modular and relatively simple to install so that the system is usable in other environments and able to be installed.

Some problems for this project would include difficulty with managing multiple tables in a scalable matter. Data being transferred and stored could become overwhelming. Additionally, formulating an algorithm to dig through such large amounts of data to calculate the wait time could become difficult.

### 3.3 Personnel Effort Requirements

| Task | Description | Time and Assignment |
|---|---|---|
| Create Requirements | Create a list of the requirements needed for the software. | 5 hours - J.W. |
| Setup Sensor Hardware | Set up seat sensors in chairs, integrate with Raspberry Pi and transmit the data in a usable format. | 25 hours - B.M |
| Integrate Hardware with Networking | Integrate with the networking tools to grab and submit data to the network. | 20 hours - T.A., J.W. |
| Develop Mobile App | Create a mobile application for customers and employees to interface with. | 30 hours - T.A., N.S. |
| Implement Server and Database | Implement a server architecture that will store data and communicate with the sensors and app. | 25 hours - S.I., N.C. |
| Test Data | Create a program to generate bogus data for use later in testing. | 10 hours - N.C., J.W |
| Design Prediction Algorithm | Create a basic data analytics algorithm. | 15 hours - S.I., N.S. |
| Test and Improve Prediction Algorithm | Run algorithm through data to improve algorithm. Test accuracy of the algorithm. Possible machine learning. | 20 hours - S.I., N.S. |

Table 7: Personnel Effort Requirements

### 3.4 Other Resource Requirements

- Tables for testing
- Chairs for testing
- Raspberrypi/Arduinos for data collection and analysis
- Transmitters to send data from tables.
- Pressure sensors to detect occupancy.

### 3.5 Financial Requirements

The financial requirements of our project are fairly minimal simply due to the research our team did when choosing our hardware components. The major expenses include the table sensor hardware which is composed of one arduino nano, one Infrared Proximity Sensor, and One Force sensor.

Multiple website were used in hardware selection process. These websites include Amazon, Digikey, Mouser, and Pololu.

Per restaurant table with 4 seats:

| Reference Number | Item | Cost per Unit x Number of Units | Total Cost |
|---|---|---|---|
| 1 | NRF24L01 Wireless RF Transceiver Module | $1.19 x 5 | $5.95 |
| 2 | Infrared Proximity Sensor | $13.95 x 4 | $55.80 |
| 3 | Force Sensor | $14.99 x 4 | $59.96 |
| 4 | Elegoo Uno | $10.96 x 1 | $10.96 |
| 5 | Elegoo Nano | $5.00 x 4 | $20 |
|  |  |  | Total: $152.67 |

Table 8: Hardware Equipment Costs per table with 4 seats

Per restaurant:

| Reference Number | Item | Cost per Unit x Number of Units | Total Cost |
|---|---|---|---|
| 1 | NRF24L01 Wireless RF Transceiver Module | $1.19 x 1 | $1.19 |
| 2 | Raspberry Pi 3 Model B | $39.00 x 1 | $39.00 |
| 3 | 16GB MircoSD | $7.34 x 1 | $7.34 |
|  |  |  | Total: $47.53 |

Table 9: Hardware Equipment Costs per restaurant

# 4 Closure Materials

## 4.1 Conclusion

Currently, inaccurate wait times cause frustration for consumers which translate to lower profits. Wait times are presently estimated by hand or very simple algorithms that is manually input by wait staff.

We believe that our design can simultaneously lower wait times and increase worker productivity which will in turn maximize customer satisfaction and maximize profits for restaurants at large.  Our design accomplishes this by eliminating the need for manual data entry and providing prediction based on customer presence and evolving data.

By simply installing our sensor nodes at each table, our central forwarding unit will stream customer driven data to our databases for analysis, the end result of which can be viewed directly by our clients on a mobile device of their choosing.  By accessing this data and using it to address their customers needs and cater their personal workflow, the entire dining process will run more smoothly for both customers and staff alike.

By strict adherence to this project plan we can effectively reduce wait times and increase customer satisfaction of diners worldwide.

## 4.2 References

(This will be built up over the course of the year)

“Arduino Nano.” *Components101*, components101.com/microcontrollers/arduino-nano.

“Raspberry Pi Hardware.” *Raspberry Pi Hardware - Raspberry Pi Documentation*, www.raspberrypi.org/documentation/hardware/raspberrypi/README.md.

Systems, eZ. “nRF24L01+.” *nRF24L01+ / 2.4GHz RF / Products / Home - Ultra Low Power Wireless Solutions from NORDIC SEMICONDUCTOR*, www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P.

## 4.3 Appendices

(This will be built up over the course of the year)